Predicting Gene Function using Predictive Clustering Trees

Celine Vens, Leander Schietgat, Jan Struyf, Hendrik Blockeel, Dragi Kocev, and Sašo Džeroski

1 Introduction

The completion of several genome projects in the past decade has generated the full genome sequence of many organisms. Identifying genes in the sequences and assigning biological functions to them has now become a key challenge in modern biology. This last step is often guided by automatic discovery processes, which interact with the laboratory experiments.

More precisely, biologists have a set of possible functions that genes may have, and these functions are organized in a hierarchy (see Fig. 1 for an example). It is known that a single gene may have multiple functions. Machine learning techniques are used to predict these gene functions. Afterwards, the predictions with highest confidence can be tested in the lab.

There are two characteristics of the function prediction task that distinguish it from common machine learning problems: (1) a single gene may have multiple functions; and (2) the functions are organized in a hierarchy: a gene that is related to some function is automatically related to all its parent functions (this is called the hierarchy constraint). This particular problem setting is known as hierarchical multi-label classification (HMC).

Several methods can be distinguished that handle HMC tasks. A first approach transforms an HMC task into a separate binary classification task for each class in the hierarchy and applies a known classification algorithm. We refer to it as the SC (single-label classification) approach. This technique

Celine Vens · Leander Schietgat · Jan Struyf · Hendrik Blockeel
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium e-mail: {Celine.Vens,Leander.Schietgat,Jan.Struyf,Hendrik.Blockeel}@cs.kuleuven.be

Dragi Kocev · Sašo Džeroski
Department of Knowledge Technologies, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia e-mail: {Dragi.Kocev,Saso.Dzeroski@ijs.si}

```
1 METABOLISM
1.1 amino acid metabolism
1.1.3 assimilation of ammonia, metabolism of the glutamate group
1.1.3.1 metabolism of glutamine
1.1.3.1.1 biosynthesis of glutamine
1.1.3.1.2 degradation of glutamine
...
1.2 nitrogen, sulfur, and selenium metabolism
...
2 ENERGY
2.1 glycolysis and gluconeogenesis
...
```

Fig. 1 A small part of the hierarchical FunCat classification scheme [34].

has several disadvantages. First, it is *inefficient*, because the learner has to be run $|C|$ times, with $|C|$ the number of classes, which can be hundreds or thousands in this application. Second, from the knowledge discovery point of view, the learned models *identify features relevant for one class*, rather than identifying features with high overall relevance. Finally, *the hierarchy constraint is not taken into account*, i.e., it is not automatically imposed that an instance belonging to a class should belong to all its superclasses.

A second approach is to adapt the SC method, so that this last issue is dealt with. Some authors have proposed to hierarchically combine the class-wise models in the prediction stage, so that a classifier constructed for a class c can only predict positive if the classifier for the parent class of c has predicted positive [4]. In addition, one can also take the hierarchy constraint into account during training by restricting the training set for the classifier for class c to those instances belonging to the parent class of c [11, 12]. This approach is called the HSC (hierarchical single-label classification) approach throughout the text.

A third approach is to develop learners that learn a single multi-label model that predicts all the classes of an example at once [16, 7]. In this way, the hierarchy constraint can be taken into account and features can be identified that are relevant to all classes. We call this the HMC approach.

In this work, we do not only consider tree structured class hierarchies, such as the example shown in Fig. 1, but also support more complex hierarchies structured as directed acyclic graphs (DAGs), where classes may have multiple parents. The latter occurs for example in the widely used Gene Ontology classification scheme [2].

Given our target application of functional genomics, we focus on decision tree methods, because they yield models that are interpretable for domain experts. Decision trees are well-known classifiers, which can handle large datasets, and produce accurate results. In Chapter **??** decision trees have been placed in the predictive clustering tree (PCT) context. We show how the three HMC approaches outlined above can be set in the PCT framework.

An experimental comparison shows that the approach that learns a single model (the HMC approach) outperforms the other approaches on all fronts: predictive performance, model size, and induction time. We show that the

results obtained by this method also outperform previously published results for predicting gene functions in *S. cerevisiae* (baker's or brewer's yeast) and *A. thaliana*. Moreover, we show that by upgrading our method to an ensemble technique, classification accuracy improves further. Throughout these comparisons, we use precision-recall curves to evaluate predictive performance, which are better suited for this type of problems than commonly used measures such as accuracy, precision and ROC curves.

The text is organized as follows. We start by discussing previous work on HMC approaches in gene function prediction in Section 2. Section 3 presents the three PCT approaches for HMC in detail. In Section 4, we describe the precision-recall based performance measures. Section 5 presents the classification schemes and datasets used in the empirical study described in Section 6 and Section 7. Finally, we conclude in Section 8.

2 Related Work

A number of HMC approaches have been proposed in the area of functional genomics. Several approaches predict functions of unannotated genes based on known functions of genes that are nearby in a functional association network or protein-protein interaction network [46, 13, 29, 15, 35, 30, 45]. These approaches are based on label propagation, whereas the focus of this work is on learning global predictive models.

Deng et al. [20] predict gene functions with Markov random fields using protein interaction data. They learn a model for each gene function separately and ignore the hierarchical relationships between the functions. Lanckriet et al. [32] represent the data by means of a kernel function and construct support vector machines for each gene function separately. They only predict top-level classes in the hierarchy. Lee et al. [33] have combined the Markov random field approach of [20] with the SVM approach of [32] by computing diffusion kernels and using them in kernel logistic regression.

Obozinski et al. [36] present a two-step approach in which SVMs are first learned independently for each gene function separately (allowing violations of the hierarchy constraint) and are then reconciliated to enforce the hierarchy constraint. Barutcuoglu et al. [4] have proposed a similar approach where unthresholded support vector machines are learned for each gene function separately (allowing violations of the hierarchy constraint) and then combined using a Bayesian network so that the predictions are consistent with the hierarchical relationships. Guan et al. [27] extend this method to an ensemble framework that is based on three classifiers: a classifier that learns a single support vector machine for each gene function, the Bayesian corrected combination of support vector machines mentioned above, and a classifier that constructs a single support vector machine per gene function and per data source and forms a Naive Bayes combination over the data

sources. Valentini and Re [48] also propose a hierarchical ensemble method that uses probabilistic support vector machines as base learners and combines the predictions by propagating the *weighted true path rule* both top-down and bottom-up through the hierarchy, which ensures consistency with the hierarchy constraint.

Rousu et al. [41] present a more direct approach that does not require a second step to make sure that the hierarchy constraint is satisfied. Their approach is based on a large margin method for structured output prediction [44, 47]. Such work defines a joint feature map $\Psi(x, y)$ over the input space X and the output space Y. In the context of HMC, the output space Y is the set of all possible subtrees of the class hierarchy. Next, it applies SVM based techniques to learn the weights w of the discriminant function $F(x, y) = \langle w, \Psi(x, y) \rangle$, with $\langle \cdot, \cdot \rangle$ the dot product. The discriminant function is then used to classify a (new) instance x as $\text{argmax}_{y \in Y} F(x, y)$. There are two main challenges that must be tackled when applying this approach to a structured output prediction problem: (a) defining Ψ, and (b) finding an efficient way to compute the argmax function (the range of this function is Y, which is of size exponential in the number of classes). Rousu et al. [41] describe a suitable Ψ and propose an efficient method based on dynamic programming to compute the argmax. Astikainen et al. [3] extend this work by applying two kernels for structured output to the prediction of enzymatic reactions.

If a domain expert is interested in knowledge that can provide insight in the biology behind the predictions, a disadvantage of using support vector machines is the lack of interpretability: it is very hard to find out why a support vector machine assigns certain classes to an example, especially if a non-linear kernel is used.

Clare [16] presents an HMC decision tree method in the context of predicting gene functions of *S. cerevisiae*. She adapts the well-known decision tree algorithm C4.5 [39] to cope with the issues introduced by the HMC task. First, where C4.5 normally uses class entropy for choosing the best split, her version uses the sum of entropies of the class variables. Second, she extends the method to predict classes on several levels of the hierarchy, assigning a larger cost to misclassifications higher up in the hierarchy. The resulting tree is transformed into a set of rules, and the best rules are selected, based on a significance test on a validation set. Note that this last step violates the hierarchy constraint, since rules predicting a class can be dropped while rules predicting its subclasses are kept. The non-hierarchical version of her method was later used to predict gene functions for *A. thaliana* [17]. Here the annotations are considered one level at the time, which also results in violations of the hierarchy constraint.

Geurts et al. [25] recently presented a decision tree based approach related to predictive clustering trees. They start from a different definition of variance and then kernelize this variance function. The result is a decision tree induction system that can be applied to structured output prediction using a method similar to the large margin methods mentioned above [47, 44]. There-

fore, this system could also be used for HMC after defining a suitable kernel. To this end, an approach similar to that of Rousu et al. [41] could be used.

Blockeel et al. [7, 5] proposed the idea of using predictive clustering trees [6] for HMC tasks. This work [7] presents the first thorough empirical comparison between an HMC and SC decision tree method in the context of tree shaped class hierarchies. Vens et al. [49] extend the algorithm towards hierarchies structured as DAGs and show that learning one decision tree for predicting all classes simultaneously, outperforms learning one tree per class (even if those trees are built taking into account the hierarchy). In Schietgat et al. [42], the predictive performance of the HMC method and ensembles thereof is compared to results reported in the biomedical literature. The latter two articles form the basis for this chapter.

3 Predictive Clustering Tree Approaches for HMC

We start this section by defining the HMC task more formally (Section 3.1). Next, we instantiate three decision tree algorithms for HMC tasks in the PCT framework: an HMC algorithm (Section 3.2), an SC algorithm (Section 3.3), and an HSC algorithm (Section 3.4).

3.1 Formal Task Description

We define the task of hierarchical multi-label classification as follows:

Given:

- an instance space X,
- a class hierarchy $(C, \leq_h)$, where C is a set of classes and $\leq_h$ is a partial order representing the superclass relationship (for all $c_1, c_2 \in C$: $c_1 \leq_h c_2$ if and only if c_1 is a superclass of c_2),
- a set T of examples (x_k, S_k) with $x_k \in X$ and $S_k \subseteq C$ such that $c \in S_k \Rightarrow \forall c' \leq_h c : c' \in S_k$, and
- a quality criterion q (which typically rewards models with high predictive accuracy and low complexity).

Find: a function $f : X \rightarrow 2^C$ (where 2^C is the power set of C) such that f maximizes q and $c \in f(x) \Rightarrow \forall c' \leq_h c : c' \in f(x)$. We call this last condition the hierarchy constraint.

In our work, the function f is represented with predictive clustering trees.

3.2 Clus-HMC: An HMC Decision Tree Learner

The approach that we present is based on decision trees and is set in the predictive clustering tree (PCT) framework [6], see Chapter **??**. This framework views a decision tree as a hierarchy of clusters: the top-node corresponds to one cluster containing all training examples, which is recursively partitioned into smaller clusters while moving down the tree. PCTs can be applied to both clustering and prediction tasks. The PCT framework is implemented in the Clus system, which is available at `http://dtai.cs.kuleuven.be/clus`.

Before explaining the approach in more detail, we show an example of a (partial) predictive clustering tree predicting the functions of *S. cerevisiae* using homology data from Clare [16] (Fig. 2). The homology features are based on a sequence similarity search for each gene in yeast against all the genes in SwissProt. The functions are taken from the FunCat classification scheme [34]. Each internal node of the tree contains a test on one of the features in the dataset. Here, the attributes are binary and have been obtained after preprocessing the relational data with a frequent pattern miner. The root node, for instance, tests whether there exists a SwissProt protein that has a high similarity (e-value $< 1.0 \cdot 10^{-8}$) with the gene under consideration G, is classified into the rhizobiaceae group and has references to the database Interpro. In order to predict the functions of a new gene, the gene is routed down the tree according to the outcome of the tests. When a leaf node is reached, the gene is assigned the functions that are stored in it. Only the most specific functions are shown in the figure. In the rest of this section, we explain how the PCT is constructed. A detailed explanation is given in Vens et al. [49].

PCTs [6] are explained in Chapter **??** and can be constructed with a standard "top-down induction of decision trees" (TDIDT) algorithm, similar to Cart [10] or C4.5 [39]. The algorithm (see Fig. **??**) takes as input a set of training instances (i.e., the genes and their annotations). It searches for the best acceptable test that can be put in a node. If such a test can be found then the algorithm creates a new internal node and calls itself recursively to construct a subtree for each subset (cluster) in the partition induced by the test on the training instances. To select the best test, the algorithm scores the tests by the reduction in variance (which is to be defined further) they induce on the instances. Maximizing variance reduction maximizes cluster homogeneity and improves predictive performance. If no acceptable test can be found, that is, if no test significantly reduces variance, then the algorithm creates a leaf and labels it with a representative case, or prototype, of the given instances.

To apply PCTs to the task of hierarchical multi-label classification, the variance and prototype are instantiated as follows [49].

First, the set of labels of each example is represented as a vector with binary components; the i'th component of the vector is 1 if the example

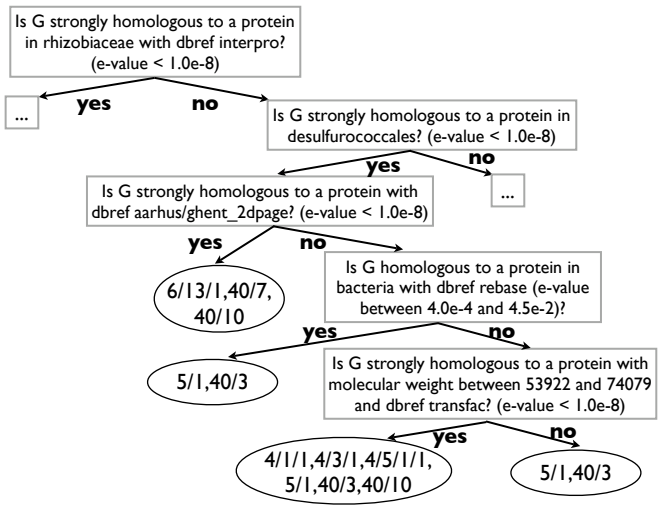

Fig. 2 Example of a predictive clustering tree, where the functions of a gene G are predicted, based on homology data.

belongs to class c_i and 0 otherwise. It is easily checked that the arithmetic mean of a set of such vectors contains as i'th component the proportion of examples of the set belonging to class c_i. We define the variance of a set of examples as the average squared distance between each example's class vector v_k and the set's mean class vector $\overline{v}$, i.e.,

$$Var(S) = \frac{\sum_k d(v_k, \overline{v})^2}{|S|}.$$

In HMC applications, it is generally considered more important to avoid making mistakes for terms at higher levels of the hierarchy than for terms at lower levels. For example in gene function prediction, predicting an "energy" gene function (i.e. FunCat class 1, see Fig. 1) while the gene is involved in "metabolism" (FunCat class 2) is worse than predicting "biosynthesis of glutamine" (FunCat class 1.1.3.1.1) instead of "degradation of glutamine" (FunCat class 1.1.3.1.2). To that aim, we use a weighted Euclidean distance

$$d(v_1, v_2) = \sqrt{\sum_i w(c_i) \cdot (v_{1,i} - v_{2,i})^2},$$

where $v_{k,i}$ is the i'th component of the class vector v_k of an instance x_k, and the class weights $w(c)$ decrease with the depth of the class in the hierarchy. We choose $w(c) = w_0 \cdot \text{avg } w(par_j(c))$, where $par_j(c)$ denotes the j'th parent of class c (the top-level classes have an artificial root class with weight $w(root) = 1$) and $0 < w_0 < 1$. Note that our definition of $w(c)$ allows the classes to be

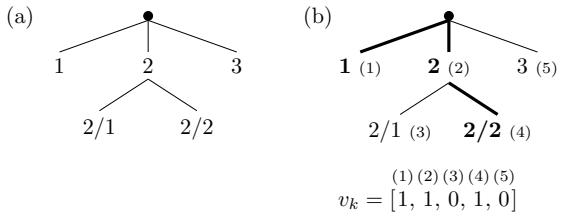

Fig. 3 (a) A toy hierarchy. Class label names reflect the position in the hierarchy, e.g., '2/1' is a subclass of '2'. (b) The set of classes {1,2,2/2}, indicated in bold in the hierarchy, and represented as a vector.

structured in a DAG, as is the case with the Gene Ontology. Consider for example the class hierarchy shown in Fig. 3, and two examples (x_1, S_1) and (x_2, S_2) with $S_1 = \{1, 2, 2/2\}$ and $S_2 = \{2\}$. Using a vector representation with consecutive components representing membership of class 1, 2, 2/1, 2/2 and 3, in that order,

$$d([1, 1, 0, 1, 0], [0, 1, 0, 0, 0]) = \sqrt{w_0 + w_0^2}.$$

The heuristic for choosing the best test for a node of the tree is then maximization of the variance reduction as discussed before, with the above definition of variance.

Second, a classification tree stores in a leaf the majority class for that leaf; this class will be the tree's prediction for examples arriving in the leaf. But in our case, since an example may have multiple classes, the notion of "majority class" does not apply in a straightforward manner. Instead, the mean $\bar{v}$ of the class vectors of the examples in that leaf is stored. Recall that $\bar{v}_i$ is the proportion of examples in the leaf belonging to c_i. An example arriving in the leaf can therefore be predicted to belong to class c_i if $\bar{v}_i$ is above some threshold t_i, which can be chosen by a domain expert. To ensure that the predictions fulfil the hierarchy constraint (whenever a class is predicted its superclasses are also predicted), it suffices to choose $t_i \leq t_j$ whenever c_i is a superclass of c_j. The PCT that is shown in Fig. 2 has a threshold of $t_i = 0.4$ for all i.

We call the resulting instantiation of the PCT algorithm in the CLUS system CLUS-HMC.

3.3 Clus-SC: Learning a Separate Tree for Each Class

The second approach that we consider builds a separate tree for each class in the hierarchy. Each of these trees is a single-label binary classification tree. Assume that the tree learner takes as input a set of examples labeled positive

or negative. To construct the tree for class c with such a learner, we label the class c examples positive and all the other examples negative. The resulting tree predicts the probability that a new instance belongs to c. We refer to this method as single-label classification (SC).

In order to classify a new instance, SC thresholds the predictions of the different single-label trees, similar to CLUS-HMC. Note, however, that this does not guarantee that the hierarchy constraint holds, even if the thresholds are chosen such that $t_i \leq t_j$ whenever $c_i \leq_h c_j$.

The class-wise trees can be constructed with any classification tree induction algorithm. Note that CLUS-HMC reduces to a single-label binary classification tree learner when applied to such data; its class vector then reduces to a single component and its heuristic reduces to CART's Gini index [10]. We can therefore use the same induction algorithm (CLUS-HMC) for both the HMC and SC approaches. This makes the results easier to interpret. It has been confirmed [7] that on binary classification tasks, CLUS-HMC performs comparably to state-of-the-art decision tree learners. We call the SC approach with CLUS-HMC as decision tree learner CLUS-SC.

3.4 Clus-HSC: Learning a Separate Tree for Each Hierarchy Edge

Building a separate decision tree for each class has several disadvantages, such as the possibility of violating the hierarchy constraint. In order to deal with this issue, the CLUS-SC algorithm can be adapted as follows.

For a non top-level class c in a tree structured hierarchy, it holds that an instance can only belong to c if it belongs to c's parent $par(c)$. An alternative approach to learning a tree that directly predicts c, is therefore to learn a tree that predicts c given that the instance belongs to $par(c)$. Learning such a tree requires fewer training instances: only the instances belonging to $par(c)$ are relevant. The subset of these instances that also belong to c become the positive instances and the other instances (those belonging to $par(c)$ but not to c) the negative instances. The resulting tree predicts the conditional probability $P(c|par(c))$. W.r.t. the top-level classes, the approach is identical to CLUS-SC, i.e., all training instances are used.

To make predictions for a new instance, we use the product rule $P(c) = P(c \mid par(c)) \cdot P(par(c))$ (for non top-level classes). This rule applies the trees recursively, starting from the tree for a top-level class. For example, to compute the probability that the instance belongs to class 2.2, we first use the tree for class 2 to predict $P(2)$ and next the tree for class 2.2 to predict $P(2.2 \mid 2)$. The resulting probability is then $P(2.2) = P(2.2 \mid 2) \cdot P(2)$. For DAG structured hierarchies, the product rule can be applied for each parent class separately, and will yield a valid estimate of $P(c)$ based on that parent. To obtain an estimate of $P(c)$ based on all parent classes, we aggregate

over the parent-wise estimates. In order to fulfil the hierarchy constraint, we use as aggregate function the minimum of the parent-wise estimates, i.e., $P(c) = \min_j P(c \mid par_j(c)) \cdot P(par_j(c))$.

Again, these probabilities are thresholded to obtain the predicted set of classes. As with Clus-HMC, to ensure that this set fulfills the hierarchy constraint, it suffices to choose a threshold $t_i \leq t_j$ whenever $c_i \leq_h c_j$. We call the resulting algorithm Clus-HSC (hierarchical single-label classification).

3.5 Ensembles of Predictive Clustering Trees

Ensemble methods are learning methods that construct a set of classifiers for a given prediction task and classify new examples by combining the predictions of each classifier. In this chapter, we consider bagging, an ensemble learning technique that has primarily been used in the context of decision trees.

Bagging [8] is an ensemble method where the different classifiers are constructed by making bootstrap replicates of the training set and using each of these replicates to construct one classifier. Each bootstrap sample is obtained by randomly sampling training instances, with replacement, from the original training set, until an equal number of instances is obtained. The individual predictions given by each classifier can be combined by taking the average (for numeric targets) or the majority vote (for nominal targets). Breiman [8] has shown that bagging can give substantial gains in predictive performance of decision tree learners. Also in the case of learning PCTs for predicting multiple targets at once, decision tree methods benefit from the application of bagging [31]. However, it is clear that, by using bagging on top of the PCT algorithm, the learning time of the model increases significantly, resulting in a clear trade-off between predictive performance and efficiency to be considered by the user.

The algorithm for bagging the PCTs takes an extra parameter k as input that denotes the number of trees in the ensemble. In order to make predictions, the average of all class vectors predicted by the k trees in the ensemble is computed, and then the threshold is applied as before. This ensures that the hierarchy constraint holds.

In the experiments, we will use bagged Clus-HMC trees. We call the resulting instantiation of the bagging algorithm around the Clus-HMC algorithm Clus-HMC-Ens.

4 Evaluation Measure

We will report our predictive performance results with precision-recall curves. Precision is the probability that a positive prediction is correct, and recall is

the probability that a positive instance is predicted positive. Remember that every leaf in the tree contains a vector $\bar{v}$ with for each class the probability that the instance has this class. When decreasing CLUS-HMC's prediction threshold t_i from 1 to 0, an increasing number of instances is predicted as belonging to class c_i, causing the recall for c_i to increase whereas precision may increase or decrease (with normally a tendency to decrease). Thus, a tree with specified threshold has a single precision and recall, and by varying the threshold a precision-recall curve (PR curve) is obtained. Such curves allow us to evaluate the predictive performance of a model regardless of t. In the end, a domain expert can choose a threshold according to the point on the curve which is most interesting to him.

Our decision to conduct a precision-recall based evaluation is motivated by the following three observations: (1) precision-recall evaluation was used in earlier approaches to gene function prediction [20, 15], (2) it allows one to simultaneously compare classifiers for different classification thresholds, and (3) it suits the characteristics of typical HMC datasets, in which many classes are infrequent (i.e., typically only a few genes have a particular function). Viewed as a binary classification task for each class, this implies that for most classes the number of negative instances by far exceeds the number of positive instances. We are more interested in recognizing the positive instances (i.e., that a gene has a given function), rather than correctly predicting the negative ones (i.e., that a gene does not have a particular function). ROC curves [38] are less suited for this task, exactly because they reward a learner if it correctly predicts negative instances (giving rise to a low false positive rate). This can present an overly optimistic view of the algorithm's performance [19].

Although a PR curve helps in understanding the predictive behavior of the model, a single performance score is more useful to compare models. A score often used to this end is the area between the PR curve and the recall axis, the so-called "area under the PR curve" (AUPRC). The closer the AUPRC is to 1.0, the better the model is.

With hundreds of classes, each of which has its own PR curve, there is the question of how to evaluate the overall performance of a system. We can construct a single "average" PR curve for all classes together by transforming the multi-label problem into a binary single-label one, i.e., by counting instance-class-couples instead of instances [49]. An instance-class couple is (predicted) positive if the instance has (is predicted to have) that class, it is (predicted) negative otherwise. The definition of precision and recall is then as before. We call the corresponding area the "area under the average PR curve" (AU($\overline{\text{PRC}}$)).

5 Datasets

Gene functions are categorized into ontologies for several reasons. First, they provide biologists with a controlled vocabulary; second, they reflect biological interdependences; and third, they ease the use of computational analysis. In this work, we consider two such ontologies: the Functional Catalogue and the Gene Ontology.

The MIPS Functional Catalogue (FunCat, `http://mips.gsf.de/projects/funcat`) [34] is a tree structured vocabulary with functional descriptions of gene products, consisting of 28 main categories. A small part of it is shown in Fig.1.

The structure of the Gene Ontology (GO, `http://www.geneontology.org`) [2] scheme differs substantially from FunCat, as it is not strictly hierarchical but organized as directed acyclic graphs, i.e. it allows more than one parent term per child. Another difference of the GO architecture is that it is organized as three separate ontologies: biological process, molecular function, and cellular localization. As can be seen in Table 3, GO has much more terms than FunCat.

Next to using two different classification schemes, we predict gene functions of two organisms: *Saccharomyces cerevisiae* and *Arabidopsis thaliana*, two of biology's classic model organisms. We use datasets described in [4], [16], and [17], with different sources of data that highlight different aspects of gene function. All datasets are available at the following webpage: `http://dtai.cs.kuleuven.be/clus/hmc-ens`.

5.1 Saccharomyces cerevisiae datasets

The first dataset we use ($\mathbf{D_0}$) was described by Barutcuoglu et al. [4] and is a combination of different data sources. The input feature vector for a gene consists of pairwise interaction information, membership to colocalization locale, possession of transcription factor binding sites and results from microarray experiments, yielding a dataset with in total 5930 features. The 3465 genes are annotated with function terms from a subset of 105 nodes from the Gene Ontology's *biological process* hierarchy.

We also use the 12 yeast datasets ($\mathbf{D_1} - \mathbf{D_{12}}$) from [16] (Table 1). The datasets describe different aspects of the genes in the yeast genome. They include five types of bioinformatics data: sequence statistics, phenotype, secondary structure, homology, and expression. The different sources of data highlight different aspects of gene function. The genes are annotated with functions from the FunCat classification schemes. Only annotations from the first four levels are given.

Table 1 Saccharomyces cerevisiae data set properties: number of instances $|D|$, number of attributes $|A|$.

Data set	$\|D\|$	$\|A\|$	Data set	$\|D\|$	$\|A\|$
D_1 Sequence [16] (seq)	3932	478	D_7 DeRisi et al. [21] (derisi)	3733	63
D_2 Phenotype [16] (pheno)	1592	69	D_8 Eisen et al. [22] (eisen)	2425	79
D_3 Secondary structure [16] (struc)	3851	19628	D_9 Gasch et al. [24] (gasch1)	3773	173
D_4 Homology search [16] (hom)	3867	47034	D_{10} Gasch et al. [23] (gasch2)	3788	52
D_5 Spellman et al. [43] (cellcycle)	3766	77	D_{11} Chu et al. [14] (spo)	3711	80
D_6 Roth et al. [40] (church)	3764	27	D_{12} All microarray [16] (expr)	3788	551

D_1 (seq) records sequence statistics that depend on the amino acid sequence of the protein for which the gene codes. These include amino acid frequency ratios, sequence length, molecular weight and hydrophobicity.

D_2 (pheno) contains phenotype data, which represents the growth or lack of growth of knock-out mutants that are missing the gene in question. The gene is removed or disabled and the resulting organism is grown with a variety of media to determine what the modified organism might be sensitive or resistant to.

D_3 (struc) stores features computed from the secondary structure of the yeast proteins. The secondary structure is not known for all yeast genes; however, it can be predicted from the protein sequence with reasonable accuracy, using Prof [37]. Due to the relational nature of secondary structure data, Clare performed a preprocessing step of relational frequent pattern mining; D_3 includes the constructed patterns as binary attributes.

D_4 (hom) includes for each yeast gene, information from other, homologous genes. Homology is usually determined by sequence similarity; here, PSI-BLAST [1] was used to compare yeast genes both with other yeast genes and with all genes indexed in SwissProt v39. This provided for each yeast gene a list of homologous genes. For each of these, various properties were extracted (keywords, sequence length, names of databases they are listed in, ...). Clare preprocessed this data in a similar way as the secondary structure data to produce binary attributes.

$D_5, \ldots, D_{12}$. Many microarray datasets exist for yeast and several of these were used [16]. Attributes for these datasets are real valued, representing fold changes in expression levels.

5.2 Arabidopsis thaliana datasets

We use six datasets from [17] (Table 2), originating from different sources: sequence statistics, expression, predicted SCOP class, predicted secondary structure, InterPro and homology. Each dataset comes in two versions: with

Table 2 Arabidopsis thaliana data set properties: number of instances $|D|$, number of attributes $|A|$.

Data set	$\|D\|$	$\|A\|$	Data set	$\|D\|$	$\|A\|$
D_{13} Sequence (seq)	3719	4450	D_{14} Expression (exprindiv)	3496	1251
D_{15} SCOP superfamily (scop)	3097	2003	D_{16} Secondary structure (struc)	3719	14804
D_{17} InterProScan data (interpro)	3719	2815	D_{18} Homology search (hom)	3473	72870

annotations from the FunCat classification scheme and from the Gene Ontology's *molecular function* hierarchy. Again, only annotations for the first four levels are given. We use the manual annotations for both schemes.

D_{13} (seq) records sequence statistics in exactly the same way as for *S. cerevisiae*. **D_{14}** (exprindiv) contains 43 experiments from NASC's Affymetrix service "Affywatch" (`http://affymetrix.arabidopsis.info/AffyWatch.html`), taking the signal, detection call and detection p-values. **D_{15}** (scop) consists of SCOP superfamily class predictions made by the Superfamily server [26]. **D_{16}** (struc) was obtained in the same way as for *S. cerevisiae*. **D_{17}** (interpro) includes features from several motif or signature finding databases, like PROSITE, PRINTS, Pfam, ProDom, SMART and TIGRFAMs, calculated using the EBI's stand-alone InterProScan package [51]. To obtain features, the relational data was mined in the same manner as the structure data. **D_{18}** (hom) was obtained in the same way as for *S. cerevisiae*, but now using SWISSPROT v41.

6 Comparison of Clus-HMC/SC/HSC

In order to compare the three PCT approaches for HMC tasks, we use the 12 yeast data sets **D_1** to **D_{12}** from Clare [16], but with new and updated class labels. We construct two versions of each data set. The input attributes are identical in both versions, but the classes are taken from the two different classification schemes FunCat and GO (we use GO's "is-a" relationship between terms). GO has an order of magnitude more classes than FunCat for our data sets: the FunCat datasets have 1362 classes on average, spread over 6 levels, while the GO datasets have 3997 classes, spread over 14 levels, see Table 3. The 24 resulting datasets can be found on the following webpage: `http://dtai.cs.kuleuven.be/clus/hmcdatasets.html`.

Clus-HMC was run as follows. For the weights used in the weighted Euclidean distance in the variance calculation, w_0 was set to 0.75. The minimal number of examples a leaf has to cover was set to 5. The F-test stopping criterion takes a "significance level" parameter s, which was optimized as follows: for each out of 6 available values for s, Clus-HMC was run on 2/3 of the training set and its PR curve for the remaining 1/3 validation set was

Table 3 Properties of the two classification schemes for the updated yeast datasets. $|C|$ is the average number of classes actually used in the data sets (out of the total number of classes defined by the scheme). $|S|$ is the average number of labels per example, with between parentheses the average number counting only the most specific classes of an example.

	FunCat	GO
Scheme version	2.1 (2007/01/09)	1.2 (2007/04/11)
Yeast annotations	2007/03/16	2007/04/07
Total classes	1362	22960
Data set average $\|C\|$	492 (6 levels)	3997 (14 levels)
Data set average $\|S\|$	8.8 (3.2 most spec.)	35.0 (5.0 most spec.)

constructed. The s parameter yielding the largest area under this average validation PR curve was then used to train the model on the complete training set. The results for Clus-SC and Clus-HSC were obtained in the same way as for Clus-HMC, but with a separate run for each class (including separate optimization of s for each class).

Each algorithm was trained on 2/3 of each data set and tested on the remaining 1/3.

Table 4 shows the AU($\overline{\text{PRC}}$) of the three decision tree algorithms. Table 5 shows summarizing Wilcoxon outcomes comparing the AU($\overline{\text{PRC}}$) of Clus-HMC to Clus-SC and Clus-HSC[1]. We see that Clus-HMC performs better than Clus-SC and Clus-HSC, both for FunCat and GO. We see also that Clus-HSC performs better than Clus-SC on FunCat and on GO.

Table 6 shows the average number of leaves in the trees. We see that the SC trees are smaller than the HMC trees, because they each model only one class. Nevertheless, the total size of all SC trees is on average a factor 398 (FunCat) and 1392 (GO) larger than the corresponding HMC tree. This difference is bigger for GO than for FunCat because GO has an order of magnitude more classes and therefore also an order of magnitude more SC trees. Comparing HMC to HSC yields similar conclusions.

Observe that the HSC trees are smaller than the SC trees. We see two reasons for this. First, HSC trees encode less knowledge than SC ones because they are conditioned on their parent class. That is, if a given feature subset

[1] Given a pair of methods X and Y, the input to the Wilcoxon test is the test set performance (AUPRC) of the two methods on the 12 data sets. The null-hypothesis is that the median of the performance difference $Z_i = Y_i - X_i$ is zero. Briefly, the test orders the Z_i values by absolute value and then assigns them integer ranks such that the smallest $|Z_i|$ is ranked 1. It then computes the rank sum of the positive (W^+) and negative (W^-) Z_i. If $W^+ > W^-$, then Y is better than X because the distribution of Z is skewed to the right. Let $S = \min(W^+, W^-)$. The p-value of the test is the probability of obtaining a sum of ranks (W^+ or W^-) smaller than or equal to S, given that the null-hypothesis is true. In the results, we report the p-value together with W^+ and W^-.

Table 4 Predictive performance (AU($\overline{\text{PRC}}$)) of Clus-HMC, Clus-SC and Clus-HSC.

	FunCat labels			GO labels		
Data set	HMC	HSC	SC	HMC	HSC	SC
seq	0.211	0.091	0.095	0.386	0.282	0.197
pheno	0.160	0.152	0.149	0.337	0.416	0.316
struc	0.181	0.118	0.114	0.358	0.353	0.228
hom	0.254	0.155	0.153	0.401	0.353	0.252
cellcycle	0.172	0.111	0.106	0.357	0.371	0.252
church	0.170	0.131	0.128	0.348	0.397	0.289
derisi	0.175	0.094	0.089	0.355	0.349	0.218
eisen	0.204	0.127	0.132	0.380	0.365	0.270
gasch1	0.205	0.106	0.104	0.371	0.351	0.239
gasch2	0.195	0.121	0.119	0.365	0.378	0.267
spo	0.186	0.103	0.098	0.352	0.371	0.213
expr	0.210	0.127	0.123	0.368	0.351	0.249
Average:	0.194	0.120	0.118	0.365	0.361	0.249

Table 5 Comparison of the AU($\overline{\text{PRC}}$) of Clus-HMC, Clus-SC and Clus-HSC. A '$\oplus$' ('$\ominus$') means that the first method performs better (worse) than the second method according to the Wilcoxon signed rank test. The table indicates the rank sums and corresponding p-values. Differences significant at the 0.01 level are indicated in bold.

	HMC vs. SC		HMC vs. HSC		HSC vs. SC	
	Score	p	Score	p	Score	p
FunCat	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ 62/16	$7.7 \cdot 10^{-2}$
GO	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ 43/35	$7.9 \cdot 10^{-1}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$

Table 6 Average tree size (number of tree leaves) for FunCat and GO datasets. For Clus-SC and Clus-HSC we report both the total number of leaves in the collection of trees, and the average number of leaves per tree.

	Clus-HMC	Clus-SC		Clus-HSC	
		Total	Average	Total	Average
FunCat	19.8	7878	15.9	3628	7.3
GO	22.2	30908	7.6	16988	3.0

is relevant to all classes in a sub-lattice of the hierarchy, then Clus-SC must include this subset in each tree of the sub-lattice, while Clus-HSC only needs them in the trees for the sub-lattice's most general border. Second, HSC trees use fewer training examples than SC trees, and tree size typically grows with training set size.

We also measure the total induction time for all methods. Clus-HMC requires on average 3.3 (FunCat) and 24.4 (GO) minutes to build a tree.

CLUS-SC is a factor 58.6 (FunCat) and 129.0 (GO) slower than CLUS-HMC. CLUS-HSC is faster than CLUS-SC, but still a factor 6.3 (FunCat) and 55.9 (GO) slower than CLUS-HMC.

7 Comparison of (Ensembles of) Clus-HMC to State-of-the-art Methods

7.1 Comparison of Clus-HMC *to Decision Tree based Approaches*

The previous section clearly showed the superiority of CLUS-HMC over CLUS-HSC and CLUS-SC. We now investigate how this method performs compared to state-of-the-art decision tree methods for functional genomics. As explained in Section 2, Clare [16] has presented an adaptation of the C4.5 decision tree algorithm towards HMC tasks. We compare our results to the results reported by Clare and King [18] on *S. cerevisiae* ($\mathbf{D_1}$ to $\mathbf{D_{12}}$), and by Clare et al. [17] on *A. thaliana* $\mathbf{D_{13}}$ to $\mathbf{D_{18}}$. The datasets that we use in this evaluation are exactly those datasets that are used in the mentioned articles. For the 18 datasets that are annotated with FunCat classes, we will compare to the hierarchical extension of C4.5 [18], which we will refer to as C4.5H. For the 6 datasets with GO annotations, we will use the non-hierarchical version [17], as C4.5H cannot handle hierarchies structured as a DAG. We refer to this system as C4.5M. For CLUS-HMC, all parameters were set as in the previous experiment.

For evaluating their systems, Clare et al. [17] report average precision. Indeed, as the biological experiments required to validate the learned rules are costly, it is important to avoid false positives. However, precision is always traded off by recall: a classifier that predicts one example positive, but misses 1000 other positive examples may have a precision of 1, although it can hardly be called a good classifier. Therefore, we also computed the average recall of the models obtained by C4.5H/M. These models were presented as rules derived from the trees, which enables us to plot only one point in PR space.

For each of the datasets these PR points are plotted against the average PR curves for CLUS-HMC. As we are comparing curves with points, we speak of a "win" for CLUS-HMC when its curve is above C4.5H/M's point, and of a "loss" when it is below the point. Under the null hypothesis that both systems perform equally well, we expect as many wins as losses. We observed that only in one case out of 24, C4.5H/M outperforms CLUS-HMC. For all other cases there is a clear win for CLUS-HMC. Representative PR curves can be found in Fig. 4 (left) and 5. For each of these datasets, we also compared the precision of C4.5H/M and CLUS-HMC, at the recall obtained by C4.5H/M. The results can be found in Fig. 6. The average gain in precision

w.r.t. C4.5H/M is 0.209 for Clus-HMC. Note that these figures also contain the results for the ensemble version of Clus-HMC (see further).

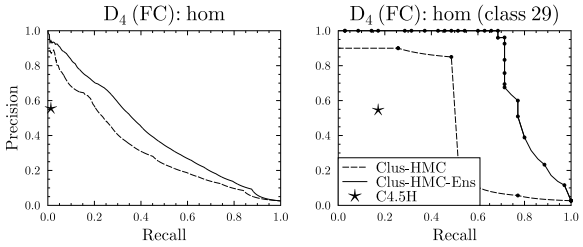

Fig. 4 Left: Average precision/recall over all classes for C4.5H, Clus-HMC and Clus-HMC-Ens on D_4 with FunCat annotations. Right: Precision-recall curve for class 29 on D_4 with FunCat annotations.

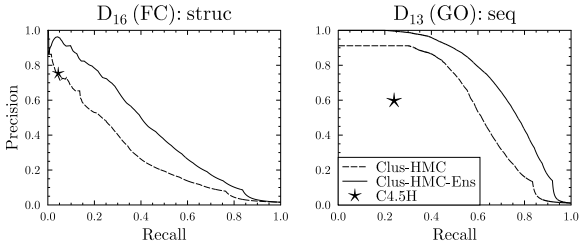

Fig. 5 Left: Average precision/recall over all classes for C4.5H, Clus-HMC and Clus-HMC-Ens on D_{16} with FunCat annotations. Right: Average curve for C4.5M, Clus-HMC and Clus-HMC-Ens on D_{13} with GO annotations.

Every leaf of a decision tree corresponds to an *if ... then ...* rule. When comparing the interpretability and precision/recall of these individual rules, Clus-HMC also performs well. For instance, take FunCat class 29, with a prior frequency of 3%. Figure 4 (right) shows the PR evaluation for the algorithms for this class using homology dataset $\mathbf{D_4}$. The PR point for C4.5H corresponds to one rule, shown in Fig. 7. This rule has a precision/recall of 0.55/0.17. Clus-HMC's most precise rule for 29 is shown in Fig. 8. This rule has a precision/recall of 0.90/0.26.

We can conclude that if interpretable models are to be obtained, Clus-HMC is the system that yields the best predictive performance. Compared with other existing methods, we are able to obtain the same precision with higher recall, or the same recall with higher precision. Moreover, the hierarchy constraint is always fulfilled, which is not the case for C4.5H/M.

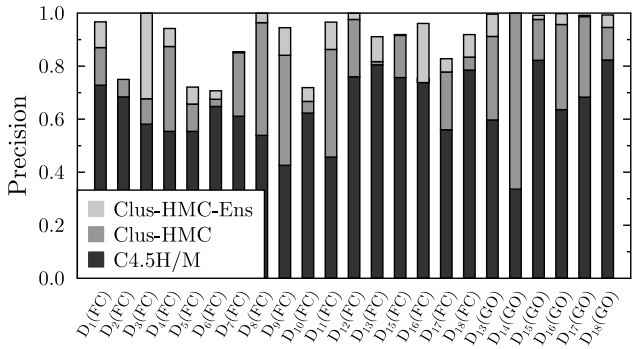

Fig. 6 Precision of the C4.5H/M, Clus-HMC and Clus-HMC-Ens algorithms, at the recall obtained by C4.5H/M on FunCat (FC) and Gene Ontology (GO) annotations. The dark grey surface represents the gain in precision obtained by Clus-HMC, the light grey surface represents the gain for Clus-HMC-Ens. D_{14}(FC) was not included, since C4.5H did not find significant rules.

if	the ORF is NOT homologous to another yeast protein ($e \geq 0.73$) and homologous to a protein in rhodospirillaceae ($e < 1.0 \cdot 10^{-8}$)
and	NOT homologous to another yeast protein ($5.0 \cdot 10^{-4} < e < 3.3 \cdot 10^{-2}$) and homologous to a protein in anabaena ($e \geq 1.1$)
and	homologous to another yeast protein ($2.0 \cdot 10^{-7} < e < 5.0 \cdot 10^{-4}$) and homologous to a protein in beta_subdivision ($e < 1.0 \cdot 10^{-8}$)
and	NOT homologous to a protein in sinorhizobium with keyword transmembrane ($e \geq 1.1$)
and	NOT homologous to a protein in entomopoxvirinae with dbref pir ($e \geq 1.1$)
and	NOT homologous to a protein in t4-like_phages with molecular weight between 1485 and 38502 ($4.5 \cdot 10^{-2} < e < 1.1$)
and	NOT homologous to a protein in chroococcales with dbref prints ($1.0 \cdot 10^{-8} < e < 4.0 \cdot 10^{-4}$)
and	NOT homologous to a protein with sequence length between 344 and 483 and dbref tigr ($e < 1.0 \cdot 10^{-8}$)
and	homologous to a protein in beta_subdivision with sequence length between 16 and 344 ($e < 1.0 \cdot 10^{-8}$)
then	class 29/0/0/0 "transposable elements, viral and plasmid proteins"

Fig. 7 Rule found by C4.5H on the D_4 homology dataset.

7.2 Comparison of Ensembles of Clus-HMC *to an SVM based Approach*

As explained in Sect. 3.5, we have extended Clus-HMC to an ensemble induction algorithm (referred to as Clus-HMC-Ens) in order to increase its predictive performance. More precisely, we built a bagging procedure around the PCT induction algorithm, each bag containing 50 trees in all experiments. As can be seen in Figures 4, 5, and 6, the improvement in predictive

if the ORF is NOT homologous to a protein in rhizobiaceae_group with dbref interpro ($e < 1.0 \cdot 10^{-8}$)
and NOT homologous to a protein in desulfurococcales ($e < 1.0 \cdot 10^{-8}$)
and homologous to a protein in ascomycota with dbref transfac ($e < 1.0 \cdot 10^{-8}$)
and homologous to a protein in viridiplantae with sequence length ≥ 970 ($e < 1.0 \cdot 10^{-8}$)
and homologous to a protein in rhizobium with keyword plasmid ($1.0 \cdot 10^{-8} < e < 4.0 \cdot 10^{-4}$)
and homologous to a protein in nicotiana with dbref interpro ($e < 1.0 \cdot 10^{-8}$)
then class 29/0/0/0 "transposable elements, viral and plasmid proteins"

Fig. 8 Rule found by Clus-HMC on the D_4 homology dataset.

performance that is obtained by using ensembles carries over to the HMC setting.

We now compare Clus-HMC-Ens to Bayesian-corrected SVMs [4]. This method was discussed in Sect. 2, and we refer to it as BSVM.

Barutcuoglu et al. [4] have used one dataset ($\mathbf{D_0}$) to evaluate their method. It is a combination of different data sources. The input feature vector for each *S. cerevisiae* gene consists of pairwise interaction information, membership to colocalization locale, possession of transcription factor binding sites, and results from microarray experiments. The genes are annotated with function terms from a subset of 105 nodes from the Gene Ontology's *biological process* hierarchy. They report class-wise area under the ROC convex hull (AUROC) for these 105 functions. Although we have argued that precision-recall based evaluation is more suited for HMC problems, we adopt the same evaluation metric for this comparison. We also use the same evaluation method, which is based on out-of-bag estimates [9].

Fig. 9 compares the classwise out-of-bag AUROC estimates for Clus-HMC-Ens and BSVM outputs. Clus-HMC-Ens scores better on 73 of the 105 functions, while BSVM scores better on the remaining 32 cases. According to the (two-sided) Wilcoxon signed rank test [50], the performance of Clus-HMC-Ens is significantly better ($p = 4.37 \cdot 10^{-5}$).

Moreover, Clus-HMC-Ens is faster than BSVM. Run times are compared for one of the previously used datasets having annotations from Gene Ontology's complete *biological process* hierarchy (in particular, we used $\mathbf{D_{16}}$ from Sect. 7.1, which is annotated with 629 classes). Run on a cluster of AMD Opteron processors (1.8 - 2.4GHz, $\geq$2GB RAM), Clus-HMC-Ens required 34.8 hours, while SVM-light [28], which is the first step of BSVM, required 190.5 hours for learning the models (i.e., Clus-HMC-Ens is faster by a factor 5.5 in this case).

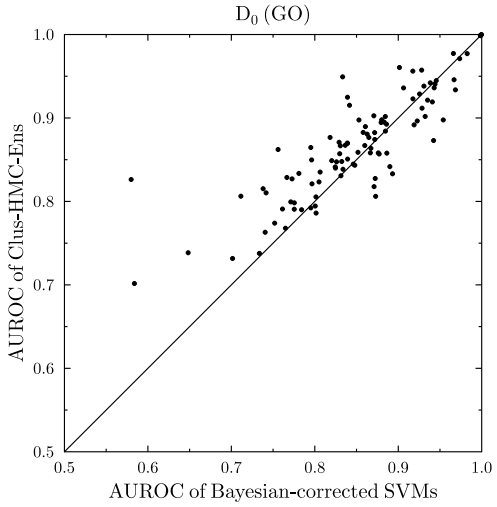

Fig. 9 Class-wise out-of-bag AUROC comparison between CLUS-HMC-ENS and Bayesian-corrected SVMs.

8 Conclusions

An important task in functional genomics is to assign a set of functions to genes. These functions are typically organized in a hierarchy: if a gene has a particular function, it automatically has its superfunctions. This setting where instances can have multiple classes and where these classes are organized in a hierarchy is called hierarchical multi-label classification (HMC) in machine learning.

In this chapter, we have presented three instantiations of the predictive clustering tree framework for HMC: (1) an algorithm that learns a single tree that predicts all classes at once (CLUS-HMC), (2) an algorithm that learns a separate decision tree for each class (CLUS-SC), and (3) an algorithm that learns and applies such single-label decision trees in a hierarchical way (CLUS-HSC). The three algorithms are designed for problems where the class hierarchy is either structured as a tree or as a directed acyclic graph (DAG).

An evaluation of these approaches on functional genomics datasets shows that CLUS-HMC outperforms the other approaches on all fronts: predictive performance, model size, and induction times. We also show that CLUS-HMC outperforms a known decision tree learner (C4.5H). Moreover, it is possible to maximize predictive performance by constructing an ensemble of CLUS-HMC-trees. We show that the latter outperforms an approach based on SVMs, while still being efficient and easy to use.

Our evaluation makes use of precision-recall curves, which give the domain expert more insight into the relation between precision and recall. We argued that PR-based evaluation measures are best suited for HMC problems, since they do not reward the negative predictions, i.e., predicting an example not to have particular labels (like ROC curves do).

We conclude that predictive clustering tree based methods are currently the most efficient, easy-to-use, and flexible approach to gene function prediction, flexible in the sense that they cover the spectrum from highly interpretable to highly accurate models.

Acknowledgements Celine Vens is a postdoctoral fellow of the Research Foundation - Flanders (FWO-Vlaanderen). Leander Schietgat is supported by a PhD grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and the ERC Starting Grant 240186: Mining Graphs and Networks: a Theory-based approach.

The authors would like to thank Amanda Clare and Zafer Barutcuoglu for providing them with the datasets and the anonimous reviewers for providing many useful suggestions. This research was conducted utilizing high performance computational resources provided by K.U.Leuven, `http://ludit.kuleuven.be/hpc`.

References

1. Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.: Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. Nucl. Acids. Res **25**, 3389–3402 (1997)
2. Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., Sherlock, G.: Gene Ontology: Tool for the unification of biology. The Gene Ontology Consortium. Nature Genetics **25**(1), 25–29 (2000)
3. Astikainen, K., L., H., Pitkanen, E., S., S., Rousu, J.: Towards structured output prediction of enzyme function. BMC Proceedings **2**(Suppl 4), S2 (2008)
4. Barutcuoglu, Z., Schapire, R., Troyanskaya, O.: Hierarchical multi-label prediction of gene function. Bioinformatics **22**(7), 830–836 (2006). DOI 10.1093/bioinformatics/btk048. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/22/7/830`
5. Blockeel, H., Bruynooghe, M., Džeroski, S., Ramon, J., Struyf, J.: Hierarchical multi-classification. In: Proceedings of the ACM SIGKDD 2002 Workshop on Multi-Relational Data Mining (MRDM 2002), pp. 21–35 (2002)
6. Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: Proceedings of the 15th International Conference on Machine Learning, pp. 55–63 (1998)
7. Blockeel, H., Schietgat, L., Struyf, J., Džeroski, S., Clare, A.: Decision trees for hierarchical multilabel classification: A case study in functional genomics. In: Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, *Lecture Notes in Artificial Intelligence*, vol. 4213, pp. 18–29 (2006)
8. Breiman, L.: Bagging predictors. Machine Learning **24**(2), 123–140 (1996)
9. Breiman, L.: Out-of-bag estimation (1996).

10. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth, Belmont (1984)
11. Cesa-Bianchi, N., Gentile, C., Zaniboni, L.: Incremental algorithms for hierarchical classification. Journal of Machine Learning Research **7**, 31–54 (2006)
12. Cesa-Bianchi, N., Valentini, G.: Hierarchical cost-sensitive algorithms for genome-wide gene function prediction. JMLR: Workshop and Conference Proceedings (Machine Learning in Systems Biology) **8**, 14–29 (2010)
13. Chen, Y., Xu, D.: Global protein function annotation through mining genome-scale data in yeast saccharomyces cerevisiae. Nucleic Acids Research **32**(21), 6414–6424 (2004)
14. Chu, S., DeRisi, J., Eisen, M., Mulholland, J., Botstein, D., Brown, P., Herskowitz, I.: The transcriptional program of sporulation in budding yeast. Science **282**, 699–705 (1998)
15. Chua, H., Sung, W., Wong, L.: Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. Bioinformatics **22**(13), 1623–1630 (2006)
16. Clare, A.: Machine learning and data mining for yeast functional genomics. Ph.D. thesis, University of Wales, Aberystwyth (2003)
17. Clare, A., Karwath, A., Ougham, H., King, R.D.: Functional bioinformatics for Arabidopsis thaliana. Bioinformatics **22**(9), 1130–1136 (2006)
18. Clare, A., King, R.D.: Predicting gene function in Saccharomyces cerevisiae. Bioinformatics **19**(Suppl. 2), ii42–49 (2003). DOI 10.1093/bioinformatics/btg1058. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/19/suppl_2/ii42`
19. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 233–240 (2006)
20. Deng, M., Zhang, K., Mehta, S., Chen, T., Sun, F.: Prediction of protein function using protein-protein interaction data. In: Proceedings of the IEEE Computer Society Bioinformatics Conference, pp. 197–206. IEEE Computer Society (2002)
21. DeRisi, J., Iyer, V., Brown, P.: Exploring the metabolic and genetic control of gene expression on a genomic scale. Science **278**, 680–686 (1997)
22. Eisen, M., Spellman, P., Brown, P., Botstein, D.: Cluster analysis and display of genome-wide expression patterns. Proc. Nat. Acad. Sci. USA **95**, 14,863–14,868 (1998)
23. Gasch, A., Huang, M., Metzner, S., Botstein, D., Elledge, S., Brown, P.: Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. Mol. Biol. Cell **12**(10), 2987–3000 (2001)
24. Gasch, A., Spellman, P., Kao, C., Carmel-Harel, O., Eisen, M., Storz, G., Botstein, D., Brown, P.: Genomic expression program in the response of yeast cells to environmental changes. Mol. Biol. Cell. **11**, 4241–4257 (2000)
25. Geurts, P., Wehenkel, L., d'Alché Buc, F.: Kernelizing the output of tree-based methods. In: ICML '06: Proceedings of the 23rd international conference on Machine learning, pp. 345–352. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1143844.1143888
26. Gough, J., Karplus, K., Hughey, R., Chothia, C.: Assignment of homology to genome sequences using a library of hidden markov models that represent all proteins of known structure. Molecular Biology **313**(4), 903–919 (2001)
27. Guan, Y., Myers, C., Hess, D., Barutcuoglu, Z., Caudy, A., Troyanskaya, O.: Predicting gene function in a hierarchical context with an ensemble of classifiers. Genome Biology **9**(Suppl 1), S3 (2008)
28. Joachims, T.: Making large-scale SVM learning practical. In: B. Scholkopf, C. Burges, A. Smola (eds.) Advances in Kernel Methods - Support Vector Learning. MIT-Press (1999)

29. Karaoz, U., Murali, T., Letovsky, S., Zheng, Y., Ding, C., Cantor, C., Kasif, S.: Whole-genome annotation by using evidence integration in functional-linkage networks. Proceedings of the National Academy of Sciences **101**(9), 2888–2893 (2004)
30. Kim, W., Krumpelman, C., Marcotte, E.: Inferring mouse gene functions from genomic-scale data using a combined functional network/classification strategy. Genome Biology **9**(Suppl 1), S5 (2008)
31. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Ensembles of multi-objective decision trees. In: Proceedings of the 18th European Conference on Machine Learning, pp. 624–631. Springer (2007)
32. Lanckriet, G.R., Deng, M., Cristianini, N., Jordan, M.I., Noble, W.S.: Kernel-based data fusion and its application to protein function prediction in yeast. In: Proceedings of the Pacific Symposium on Biocomputing, pp. 300–311 (2004)
33. Lee, H., Tu, Z., Deng, M., Sun, F., Chen, T.: Diffusion kernel-based logistic regression models for protein function prediction. OMICS **10**(1), 40–55 (2006)
34. Mewes, H., Heumann, K., Kaps, A., Mayer, K., Pfeiffer, F., Stocker, S., Frishman, D.: MIPS: A database for protein sequences and complete genomes. Nucleic Acids Research **27**, 44–48 (1999)
35. Mostafavi, S., Ray, D., Warde-Farley, D., Grouios, C., Morris, Q.: GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. Genome Biology **9**(Suppl 1), S4 (2008)
36. Obozinski, G., Lanckriet, G., Grant, C., Jordan, M., Noble, W.: Consistent probabilistic outputs for protein function prediction. Genome Biology **9**(Suppl 1), S6 (2008)
37. Ouali, M., King, R.: Cascaded multiple classifiers for secondary structure prediction. Protein Science **9(6)**, 1162–76 (2000)
38. Provost, F., Fawcett, T.: Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, pp. 43–48. AAAI Press (1998)
39. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann series in Machine Learning. Morgan Kaufmann (1993)
40. Roth, F., Hughes, J., Estep, P., Church, G.: Fining DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. Nature Biotechnology **16**, 939–945 (1998)
41. Rousu, J., Saunders, C., Szedmak, S., Shawe-Taylor, J.: Kernel-based learning of hierarchical multilabel classification models. Journal of Machine Learning Research **7**, 1601–1626 (2006)
42. Schietgat, L., Vens, C., Struyf, J., Blockeel, H., Kocev, D., Džeroski, S.: Predicting gene function using hierarchical multi-label decision tree ensembles. BMC Bioinformatics **11**(2) (2010)
43. Spellman, P., Sherlock, G., Zhang, M., Iyer, V., Anders, K., Eisen, M., Brown, P., Botstein, D., Futcher, B.: Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. Molecular Biology of the Cell **9**, 3273–3297 (1998)
44. Taskar, B., Guestrin, C., Koller, D.: Max-margin Markov networks. In: Advances in Neural Information Processing Systems, vol. 16. MIT Press (2003)
45. Tian, W., Zhang, L., Tasan, M., Gibbons, F., King, O., Park, J., Wunderlich, Z., Cherry, J., Roth, F.: Combining guilt-by-association and guilt-by-profiling to predict saccharomyces cerevisiae gene function. Genome Biology **9**(Suppl 1), S7 (2008)
46. Troyanskaya, O., Dolinski, K., Owen, A., Altman, R., D., B.: A bayesian framework for combining heterogeneous data sources for gene function prediction (in saccharomyces cerevisiae). Proceedings of the National Academy of Sciences **100**(14), 8348–8353 (2003)

47. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6**, 1453–1484 (2005)
48. Valentini, G., Re, M.: Weighted true path rule: a multilabel hierarchical algorithm for gene function prediction. In: Proceedings of the First International Workshop on Learning from Multi-Label Data, pp. 133–146 (2009)
49. Vens, C., Struyf, J., Schietgat, L., Džeroski, S., Blockeel, H.: Decision trees for hierarchical multi-label classification. Machine Learning **73**(2), 185–214 (2008)
50. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
51. Zdobnov, E., Apweiler, R.: Interproscan - an integration platform for the signature-recognition methods in interpro. Bioinformatics **17**(9), 847–848 (2001)